Perl 2 – Regular Expressions, Hashes, References, Debugging
BIOC 8142  Feb 20, 2013
Bill Pearson  wrp@virginia.edu

• String matching and regular expressions
  – Matching with $str =~ m/^>/;
  – Substitution with $file =~ s/aa$/;
  – Capturing parts of string
    ($gi,$db,$acc)=($descr =~m/'|\d+\|\w+\|\w+.?\w+\')

• Working with arrays (lists)
• Hashes and hash slices
• Perl debugging – what is your program doing?
• References and dereferencing – multi-dimensional @arrays, %hashes

What should you do to reinforce the lecture material?

• Beginner’s introduction to Perl regular expressions
• Programming Perl (4th Ed) Wall et al, Safari Technical Books
  proquest.safaribooksonline.com/book/programming/perl/0596000278
  Ch. 2.9, Hashes; Ch. 5, Pattern matching
• man perlretut- regular expressions
• man perlrre - more regular expressions
• man perldbrtut - intro perl debug tutorial
• man perldebrug - complete debug reference
• man perlrref - references and data structures
Regular expressions

• used for string matching, substitution, pattern extraction
• /\>gi\|/ matches >gi|121694|sp|P20432.3|GTT1_DROME ...
• if ($line =~ m/^\>gi/) { ... } #match
• $line =~ /\>gi\|(\d+)\|/; # extract gi#
  $gi = $1;
• ($gi) = $line =~ /\>gi\|(\d+)\|/; #same
• $line =~ s/^\>(.*)$/\>$1;/; # substitution

Regular expressions (cont.)

• m/plaintext/
  m/one|two/; # alternation
  m/(one|two)|(three)/; # grouping with
  # parenthesis(capture)
• /\>gi\|(\d+)/ # ^beginning of line
  /.+ (\d+) aa$/ # $ end of line
• /a*bc/ # bc,abc,aabc, ... # repetitions
• /a?bc/ # abc, bc
• /a+bc/ # abc, aabc, ...
Regular Expressions, III

>gi|121694|sp|P20432.3|GSTT1_DROME Glutathione S-transferase 1-1

• Matching classes:
  - /gi\[(0-9)+\]([a-z]+)[A-Z][0-9a-z]*\|/  
    * [a-z] [0-9] -> class  
    * [0-9] -> negated class
  - /gi\|\d\|[a-z]\|\w\|/  
    * \d -> number [0-9]  
    * \w -> word [0-9a-zA-Z_] \W -> not a word char  
    * \s -> space [ \t\n\r\] \S -> not a space

• Capturing matches:
  - /gi\|\(\d\)+\|([a-z])\|\w\|/  
    * $1 $2 $3
  - ($gi,$db,$db_acc) =  
    $line = /gi\|\(\d\)+\|([a-z])\|\w\|/;

Regular expressions - modifiers

• m/That/i  # ignore case
• s/this/that/g  # global replacement
• m/>gi\|\(\d\)+\|([a-z]{2,3})\|\w\|\{range}\  
• s///m  # treat string as multiple lines
• s///s  # span over \n
• s/\n//gs  # remove \n in multiline entry
• s/GAATTC/$1\n/g  # break lines at EcoRI site

{ local $/ = "\n";  
  while ($entry = <INFILE>) {  
    chomp($entry);  
    $entry = s/[/>]+/;  
    # replace missing >, if necessary  
    # \A is like " (beginning of string)  
    open(OUT, ">$file$num.fa") or die $!;  
    $num++;  
    print OUT "$entry\n";  
    close(OUT);  
  }
String expressions
(with regular expressions)

- if ( /^>/gi\|/ ) { ... }
- if ( $line =~ m/^>/gi\|/ ) { ... }
- while ( $line !~ m/^>/gi\|/ ) { ... }
- Substitution:
  $new_line =~ s/\|/:/g; # without 'g', only once
- Pattern extraction:
  ($gi,$db,$db_acc) =
  ($sseqid =~ /^>(\d+)\|([a-z])\|([\w]+)/);
- split (\|\|,$sseqid)
- join (":",@fields);
- subst($string,$start,$length); # rarely used
- index($string,$query); # rarely used
- Comparison: “eq” “ne” “cmp” “lt” “gt” “le” “ge”

Working with arrays (lists) I –

my @months = qw(Jan Feb Mar Apr ... );
$months[0] == 'Jan'; $months[3]=='Apr';

- Create array:
  my @array=(); @array=(1..10); @array=qw(cat dog pirahana);
- Extract/set individual element:
  $value=$array[1]; $value=$array[$i];
  $array[0]=98.6; $array[$i]=101.4;
- Extract/set list of elements (@array slice)
  ($first, $third, $last) = @array[0,2,-1];
  @array[2,1,0] = ( 432, 4.5E-10, 'GSTM1_HUMAN')
- Unlike many languages, perl @array elements do not have a constant type; $array[0] can be a "string" while $array[1] is a number.
Working with arrays (lists) II–

```perl
my @months = qw(Jan Feb Mar Apr ... );
$months[0] == 'Jan';  $months[3]=='Apr';

• Add to array (array gets longer, at end or start)
  push @array, $value; $array[-1]==$value;
  – add to end of array
  unshift @array, $value; $array[0] == $value;
  – add to beginning, much much less common

• Remove from array (array gets shorter/smaller)
  $first_element=shift @array;  #slightly more common
  $last_element=pop @array;
  push/shift implements first in/first out queue (ticket line)
```

Working with arrays (lists) III–

```perl
• Sort array
  my @num_array = (2.48, 1.72, 2.15, 1.55);
  @num_array = sort { $a <=> $b } @num_array;
  @num_array == (1.55, 1.72, 2.15, 2.48);
  @num_array = sort { $b <=> $a } @num_array;
  @num_array == (2.48, 2.15, 1.72, 1.55);
  @str_array = qw(Bat Aardvark Dog Cat);
  @str_array = sort { $a cmp $b } @str_array;
  @str_array == ('Aardvark', 'Bat', 'Cat', 'Dog');

• Recover a subset of an array – grep
  @no_a_animal = grep { !/a/ } @str_array;
  @no_a_animal == ('Dog');

• Build new array – map {}
```
Perl Hashes –
Arrays with names, not positions

my @months = qw(Jan Feb Mar Apr ... );
$months[0] == 'Jan';  $months[3]=='Apr';
my @month_days = (31, 28, 31, 30, ...);
$month_days[1] == 28;

my %month_days=('Jan'=>31,'Feb'=>28,'Mar'=>31,'Apr'=>30,...);
$month_days{Feb}==28;

my %hash = ();  %-> hash
my $hash{hash_key} = value;
for my $entry ( keys(%hash) )
    { print "$entry : $hash{$entry}\n";}

#!/usr/bin/perl –w
use strict;

my @months = qw(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec);
my @month_days = (31, 28, 31, 30, 31, 30, 31, 31, 31, 31, 30, 31);

my %month_hash = ();
for (my $i=0; $i<scalar(@months); $i++) {
    $month_hash{$months[$i]} = $month_days[$i];
}
for my $month ( @months ) {print "$month\n";}
for my $month ( @months ) {
    print "$month: $month_hash{$month}\n";
}

my %month_hash2 = ();
@month_hash2(@months) = @month_days;
for my $month ( @months ) {
    print "$month: $month_hash2{$month}\n";
}
Perl Hashes (cont.)

- A hash key need not have a value
  
  ```perl
  defined($month_day{Meb}) == 0;
  $month_day{Meb}=0;
  defined($month_day{Meb}) == 1;
  ```

- `defined()` is convenient for checking for duplicates, e.g.
  
  ```perl
  if (defined($accs{P09488})) { do something;}
  else {$accs{P09488}=$evalue;} # now it is defined
  ```

- Unlike an `@array`, a `%hash` is unordered:
  
  ```perl
  for my $month (@months) {prints months in order;}
  ```
  
  ```perl
  for my $month ( keys(%months))
  { could be Dec, Mar, Sep, etc.}
  ```

  If you need the elements of a hash in order, either keep a separate array (@months), or make a 2-D hash with an index (see next)

---

Array slices / Hash slices

<table>
<thead>
<tr>
<th>qseqid</th>
<th>sseqid</th>
<th>pident</th>
<th>len</th>
<th>mis</th>
<th>sp</th>
<th>qse</th>
<th>qse</th>
<th>qse</th>
<th>qse</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp</td>
<td>GSTM1_HUMAN</td>
<td>sp</td>
<td>GSTM1_HUMAN</td>
<td>100.00</td>
<td>218</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>218</td>
</tr>
<tr>
<td>sp</td>
<td>GSTM1_HUMAN</td>
<td>sp</td>
<td>GSTM4_HUMAN</td>
<td>86.70</td>
<td>218</td>
<td>29</td>
<td>0</td>
<td>1</td>
<td>218</td>
</tr>
</tbody>
</table>

Perl loves `@arrays (@lists)`. Many perl programs NEVER refer to individual data elements with and index (no `$array[$i]`). How to easily isolate the information desired (sseqid, `evalue`)?

How do we refer to the data?

```perl
@data = split(/	/,$line);
```

1) **Array slice:**

```perl
my @hit_data = @data[1,10];
@hit_data = @data[1,-2];
```

"Slices" isolate a (possibly non-contiguous) part of an `@array[3,1,10]` or `@hash[('bits','evalue','percid')]"
Array slices / Hash slices

<table>
<thead>
<tr>
<th>qseqid</th>
<th>sseqid</th>
<th>pident</th>
<th>len</th>
<th>mis</th>
<th>gp</th>
<th>qs</th>
<th>se</th>
<th>ss</th>
<th>evalue</th>
<th>bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp</td>
<td>GSTM1_HUMAN</td>
<td>sp</td>
<td>GSTM1_HUMAN</td>
<td>100.00</td>
<td>218</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>218</td>
<td>1</td>
</tr>
<tr>
<td>sp</td>
<td>GSTM1_HUMAN</td>
<td>sp</td>
<td>GSTM4_HUMAN</td>
<td>86.70</td>
<td>218</td>
<td>29</td>
<td>0</td>
<td>1</td>
<td>218</td>
<td>1</td>
</tr>
</tbody>
</table>

```perl
@data = split(/\t/,$line);
my @hit_data = @data[1,10];
```

The problem with arrays is that you need to remember where the data is. Is $data[10] the evalue, or the bit score?

2) Hash:

```perl
my @hit_hash = ();
@hit_hash{qw(qseqid sseqid ... evalue bits)} = @data;
or
@field_names = qw(qseqid sseqid ... evalue bits);
@hit_hash{@field_names} = @data;
@hit_hash{@field_names} = split(/\t/,$line)
```

print join("\t",($hit_hash{sseqid},$hit_hash{evalue}),"\n";
```

Hash slices are confusing (why isn't it an array?)

```perl
@field_names = qw(qseqid sseqid ... evalue bits);
while (my $line = <>) {
    chomp $line;
    @hit_hash{@field_names} = split(/\t/,$line);
}
```

Why isn't @hit_hash{} an array??

Because the '@' indicates a list usage, but the '{ }' indicates its type.

(Same for @data[1,3,5], which takes a set of elements from @data and returns them as a list, so the '@' says a list of things, and the '{ }' says they are coming from an array.)

Hash slices replace:
```perl
$month_hash{Jan}=31; $month_hash{Feb}=28; ...
for (my $i=0; $i<scalar(@months); $i++) {
    $month_hash{$months[$i]} = $month_days[$i];
}
```
Perl debugging

1. Fix syntax errors (undeclared variables, missing ';', or '{'}
   perl –c script_name.pl
2. If the program does not work (or prints nonsense), or if you just want to watch it work.
   perl –d script_name.pl
   – 'n' : next
   – 'x' : examine
   – 'b' : break
   – 'c' : continue
   – 'q' : quit
   – 'h' : help
3. The debugger is a perl interpreter, so you can try anything you like.
   DB<5> x split(/s+/, "this is a short string")
   0 'this'
   1 'i'
   2 'a'
   3 'hort'
   4 'tring'
   DB<6> x split(/\s+/, "this is a short string")
   0 'this'
   1 'is'
   2 'a'
   3 'short'
   4 'string'

#!/usr/bin/perl –w
use strict;

my @months = qw(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec);
my @month_days = (31, 28, 31, 30, 31, 30, 31, 31, 31, 30, 31);

my %month_hash = ();

for (my $i=0; $i<scalar(@months); $i++) {
    $month_hash{$months[$i]} = $month_days[$i];
}

for my $month (@months ) {print "$month\n";}
for my $month (@months ) {
    print "$month: $month_hash{$month}\n";
}

my %month_hash2 = ();
@month_hash2{@months} = @month_days;

for my $month (@months ) {
    print "$month: $month_hash2{$month}\n";
}
franklin $ perl -d hash_intro.pl
Loading DB routines from perl5db.pl version 1.32
Editor support available.
Enter h or 'h h' for help, or 'man perldebug' for more help.

main::(hash_intro.pl:5): my @months = qw(Jan Feb Mar ... Oct Nov Dec);
DB<1> n
main::(hash_intro.pl:7): my @month_days = {31, 28, 31, ..., 31, 30, 31};
DB<1> n
main::(hash_intro.pl:9): my %month_hash = ();
DB<1> x $months
0 undef
DB<2> x @months
0 'Jan'
1 'Feb'
2 'Mar'
3 'Apr'
...
DB<1> n
main::(hash_intro.pl:13): } # jumps to end of loop first
DB<4> n
main::(hash_intro.pl:11): for (my $i=0; $i<scalar(@months); $i++) {
DB<4>
main::(hash_intro.pl:12): $month_hash{$months[$i]} = $month_days[$i];
DB<4>
main::(hash_intro.pl:12): $month_hash{$months[$i]} = $month_days[$i];
DB<4> x %month_hash
0 'Jan'
1 31

main::(hash_intro.pl:25): my %month_hash2 = ();
DB<8>
main::(hash_intro.pl:27): %month_hash2{@months} = @month_days;
DB<8> x %month_hash2
empty array
DB<9> x @month_hash2
empty array
DB<10> n
main::(hash_intro.pl:29): for my $month (@months) {
DB<10> x %month_hash2
0 'Sep'
1 31
2 'May'
3 31
4 'Jul'
5 31
Arrays of arrays (and hashes of hashes)

Variable dereferencing

Perl @arrays and (%hashes) are always one-dimensional, but data is usually (at least) two-dimensional.

How do we build data structures that have multiple dimensions?

hit[1]{percid} == 86.70

Variable dereferencing

To build multi-dimensional (complex) data structures, perl provides variable references, which are scalar (simple) values:

\$string_ref, \@array_ref, \%hash_ref

\$month_name_ref = \@month_name;
@{\$month_name_ref} == @month_name;

\$month_hash_ref = \%month_hash;
@{\$month_hash_ref} == %month_hash

DB<> x \$month_hash
0 'Sep' 31
1 'May' 31
4 'Jul' 30
5 'Jun' 31
8 'Jan' 31
9 31

DB<> x \%month_hash
 HASH(0xf3c5d40)
  'Apr' => 30
  'Aug' => 31
  'Dec' => 31
  'Feb' => 28
  'Jan' => 31
  'Jul' => 31
  'Jun' => 30
  'Mar' => 31
  'May' => 31

DB<> x \$month_name
0 'Sep' 31
1 'May' 31
4 'Jul' 30
5 'Jun' 31
8 'Jan' 31
9 31

...
Variable dereferencing

Since references are scalars, they can be put into @arrays and %hashes to make 2-D structures

```
DB<10> @new_array = (\@months, \@month_days, \%month_hash);
DB<11> x @new_array
  0 ARRAY(0xfb512a0)
      0 'Jan'
      1 'Feb'
      ... 
      11 'Dec'
  1 ARRAY(0xfc361a8)
      0 31
      1 28
      ... 
      11 31
  2 HASH(0xfc35d40)
      'Apr' => 30
      'Aug' => 31
      'Dec' => 31

DB<12> x $new_array[0][4]
  0 'May'

DB<13> x $new_array[0]->[4]
  0 'May'

DB<14> x $new_array[1][4]
  0 31

DB<15> x $new_array[2]->{May}
  0 31
```

Variable dereferencing

Since references are scalars, they can be put into @arrays and %hashes to make 2-D structures

```
DB<20> $new_hash = ( 
    names=>\@months,
    length=>\@month_days,
    hash=>\%month_hash);

DB<21> x \%new_hash
  0 HASH(0x100d7798)
      'days' => ARRAY(0xfc361a8)
          0 31
          1 28
          11 31
      'hash' => HASH(0xfc35d40)
          'Apr' => 30
          'Aug' => 31
          'Sep' => 31
      'months' => ARRAY(0xfb512a0)
          0 'Jan'
          1 'Feb'
          11 'Dec'
```

```
DB<22> x $new_hash{months}[1]
  0 'Feb'

DB<23> x $new_hash{hash}{Apr}
  0 30

DB<24> x $new_hash{hash}{Mar}
  Can't locate object method "hash"

DB<25> x $month_hash{Mar}
  0 undef # because hash->{Mar}==31
```
Homework 1

1. download and uncompress a tabulated microarray dataset from GEO:
2. write a Perl program
   a. to extract the probesets having an adjusted P-value <= 0.10 (10% false-discovery rate)
   b. identify instances of genes having more than one significant probeset
   c. for each instance of such genes, identify the range of significant fold-change values
   d. report cases where the range crosses 1 (i.e. cases where some probeset(s) of a gene are up-regulated, while others are down-regulated)

Homework 2

Distantly related proteins can often be shown to be homologous by finding an intermediate protein that shares significant similarity to each of the two homologous, but not significantly similar, proteins.

1. Write a script that performs a BLASTP search against SwissProt, then, for each of the non-significant matches (evalue > 0.1), runs a search (against SwissProt) with the subject sequence, and then compares the results from the first search to the "reverse" searches, attempting to find an intermediate protein that had significant similarity to the queries in both the first and second searches.
2. (optional) Modify the script to ensure that the regions of the proteins that align in the two searches overlap.